CO1301 Games Concepts: 2018-2019
Assignment 3

Release Date: Monday 18/03/19   Hand in Date: Saturday 17/04/19 (11.59pm)

Introduction

For this assignment I want you to implement the elements of a **racing game**, including car movement, race stages and collision detection / resolution. In the "basic" version of the game the player controls a hover-car which must be driven along a short desert race-track from a starting point, over three stages to a finishing gate. The stages must be completed in sequence and the player cannot finish the race unless they have completed all of the stages.

I expect you to complete this project in your own time outside of scheduled labs.

This is an **individual** project and no group work is permitted.

Do not diverge from the assignment specification. If you do not conform to the assignment specification then you will **lose marks**. If you do want to make some addition to the game (at the advanced level) and you are unsure whether the change will break the specification then check with a tutor first. Some of the requirements at the advanced level have been left deliberately vague – use your creativity to add polish to your game – it could be the start of your professional portfolio.

Avoiding Plagiarism

- You will be held responsible if someone copies your work - unless you can demonstrate that have taken reasonable precautions against copying.
- Remember that the machines in the Games Lab have **shared drives**. You must therefore either password protect your work or else **remove your work from the hard drive** each time you finish working on it.

Learning Outcomes Assessed (see module descriptor for full list)

- Describe key games concepts, e.g. genres, terminology.
- Use a game engine to create simple computer game prototypes.
- Apply mathematical techniques for analysis and reasoning about problems

Deliverables, Submission and Assessment

- As before there are two deliverables – your **source code** and a **report**.
- Submission is electronic, via Blackboard.
  - For the source code, multiple files **are** permitted for this assignment.
    - If you create multiple files, you MUST also upload the Visual Studio project files to enable me to compile and run your project.
    - DO NOT upload compiled object files or executable files.
  - Make sure you include your name as a comment on the first line of EVERY source code file. Also include your name at the top of your report.
  - See the last page for details of what to include in your report.
- Assessment will be by demonstration in your usual lab session during the week following the hand-in deadline. **Failure to attend the lab to demonstrate equates to missing an exam.**

- **Demonstrations will take place during the week commencing 29/04/19**. You must demonstrate the code you uploaded – no tweaking!
    - **Important Note**: Your demonstration counts as part of your submission. If you are late demonstrating your assignment and do not have an extension, the usual lateness penalties will apply.

      PLEASE ALSO NOTE: The last possible date for demonstrations (with or without extensions) is Friday 3rd May 2019

      I expect you to be able to show a draft version of the assignment (to at least a "PASS" level) BEFORE the Easter vacation.

Resources

This assignment has an associated set of files. The files can be found inside "Assignment 3 models.zip" on Blackboard. The files include:

- Models

race2.x	A hover car
Checkpoint.x	A checkpoint and the finishing line
Skybox 07.x	A skybox
ground.x	The ground
TankSmall1.x	A water tank
TankSmall2.x	Another water tank
IsleStraight.x	A wall end
Wall.x	A wall section
Tribune1.x	A trap
Interstellar.x	An enemy spaceship
Cross.x	A large red cross
Flare.x	A bomb

- A pack of "extra" models to enhance your scene will also be made available.
- You may use models from other sources as well if you wish (e.g. from earlier lab exercises), however, you must upload any that you do use along with your source code when you submit your assignment.

Please remember that all of the models and textures I supply are under license and must not be passed on to any else or used for any purpose other than your University work.

Game Specification

You should implement the features described below **in order**. To be eligible for a mark within any classification, you must have completed **all** the features for **all** the previous classifications. Only when you have completed the First classification may you add ideas of your own.

For a very high mark, you need to pay attention to the **playability** of the game as well as achieving the technical requirements.

A student (Alan Raby) completed a similar assignment to this a few of years ago, and placed a video of it on YouTube: https://youtu.be/af6KhD06FcA. This is the sort of thing you should be aiming for if you want a very high mark.

IMPORTANT NOTE: The bare pass mark is 40%, however if your course is Games Development, Software Engineering or Computer Science, I consider a mark of **60%** to be satisfactory.

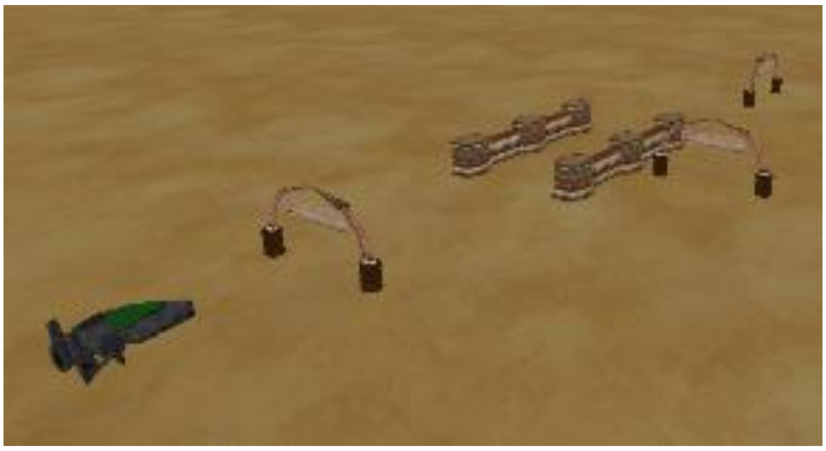
Layout of the Level

- Use the models to create a race track. The basic layout consists of three checkpoints and two walls as illustrated below.
- The walls are each made of three **Isle** models and a **Wall** model.
- The location of the models are:

    - checkpoint 1 (0, 0,   0)
    - checkpoint 2 (10, 0, 120) – *rotated 90 degrees*
    - checkpoint 3 (25, 0, 56)

    - isle (-10, 0, 40)
    - isle (10, 0, 40)
    - isle (-10, 0, 56)
    - isle (10, 0, 56)
    - isle (-10, 0, 72)
    - isle (10, 0, 72)

    - wall (-10, 0, 48)
    - wall (10, 0, 48)
    - wall (-10, 0, 64)
    - wall (10, 0, 64)



- The checkpoints, aisles and walls MUST be stored either in arrays or STL vectors.
- For very high grades, you should read the positions of scenery items in from a file.
- You are not required to use any other models unless you are attempting the higher grades, but you may use other models if you wish.
    - If you are attempting a lower second grade then you need to place some water tanks around the track
    - If you are attempting a higher second grade then you need to place more wall sections on the track.
- You may want to place some other buildings/models for effect.
- The hover-car should start some distance away from the first stage.
- The skybox should be created at location (0, -960, 0)

Bare Pass Mark = third classification (40% +) Milestone 1

- The code you submit **must** compile on a machine in the Games Lab without errors.
- Set up the scene as described above. (Note the orientation of checkpoint 2)
- Implement your own manual chase cam in which the camera is positioned above and behind the car.
  - Set up keys so that
    - 'Up' moves the camera forward
    - 'Down' moves the camera backward
    - 'Right' moves the camera right.
    - 'Left' moves the camera left.
    - Mouse movement rotates the camera. (You should not be able to "wriggle" the camera upside-down with the mouse!)
    - '1' resets the camera position

- The player can only control the hover car
- Set up keys so that
  - 'W' applies forward thrust to the hover-car
  - 'S' applies backwards thrust to the hover-car
    - The maximum backward thrust should be half that of the maximum forward thrust.
  - 'D' turns the car clockwise
  - 'A' turns the car anti-clockwise

- Use 2D vectors to control the movement of the car.
  - Calculate vectors for **thrust** (in the direction the car is **pointing**), and **drag** (in the opposite direction and proportional to the car's **momentum**).
  - Calculate the car's new **momentum** by combining the thrust and drag vectors with the previous momentum.
  - Move the car each frame according to its momentum vector.
    - You should be able to get it to slide round corners…

- You should have variables to store the thrust force and drag coefficient for use in the above calculations. Experiment with the values of these variables until you have a controllable hover car with a reasonable top speed.

- Use "ui_backdrop.jpg" to create a backdrop for dialogue and other game information. It would probably look best if this is at the bottom of the screen. Use Draw to output dialogue and any other game information.
  - You could make your game look nicer by replacing ui_backdrop.jpg with something more exciting.

- Display the current state of the game over the backdrop.
- When the game loads the dialog should read "**Hit Space to Start**".
- When the player hits start, the dialog should flash "**3**" for a second, then "**2**", then "**1**", then "**Go!**".
  - The player should not be able to move the hover-car until the dialog says "Go!"
- When the car crosses the first checkpoint the dialogue changes to "**Stage 1 complete**" and so on.
- When the car crosses the final checkpoint the dialogue changes to "**Race complete**".

- The car needs to cross the checkpoints in the correct order. You cannot complete a stage out of order.
    - You are being asked to implement **states**.
    - You could think of the race stages as sub-states within the overall game's Racing state.
- Dialogue and state changes are triggered by moving through the checkpoints.
    - You should treat the gap between the struts of each checkpoint as an axis-aligned bounding box, and implement a **point-to-box** collision function to detect the centre of the car passing through the checkpoint.

- Implement collision detection and simple resolution with the **walls**, and with the **legs of the checkpoints**.
    - You may treat the hover-car as a sphere for collision detection purposes.
    - You can treat the two isle models and wall section between them as a single axis-aligned bounding box. Implement collision detection with these as a **sphere-to-box collision**.
    - Implement a **sphere-to-sphere** calculation for the struts of the checkpoints.

- You must use **frame timing** to control the speed of EVERYTHING.
    - This includes hover car movement/steering AND camera movement
- You must use an **enumerated type** to implement the game states.
- You must use **functions** for collision detection, called from within a **loop**.
- You must declare a **structure** for a 2D vector, and implement functions to manipulate such vectors. At the very least you will need functions for scalar multiplication and vector addition.

NOTE: I must be able to TEST that your collision-driven state changes work as they should. If you make the level such that this can't be tested, you won't get the marks!

Lower second classification (50% +) Milestone 2

- Add a speed readout to the dialogue – it should output the speed in **whole** numbers, as ***kilometres per hour*** according to the **scale** you have defined for your models (*see the Report section*).
    - The scale you define in your report should be implemented as a constant in your program.
    - Multiply your momentum vector's length by your scale constant to find out the speed of your car in metres/second.

- Introduce at least one more checkpoint. You will need to extend the number of states to account for the increased number of checkpoints. You also need to add on the appropriate dialogue, e.g. "Stage 3 complete".
    - If you include any wall/isle sections, keep the walls axis-aligned (to keep collision detection simpler)

- Use water tanks placed sparingly alongside the track to suggest "corners".
- Also place a tank half-buried in the sand and leaning at an angle in the middle of the track in one of the sections as an obstacle to be avoided.
- Implement collision detection between the car and all of the stationary objects:
    - Implement collision detection between the car and the tanks as sphere-to-sphere.

- Implement a basic damage model for the player hover-car. The car starts with 100 health points. Every time the car collides with an object it should lose 1 health point. Display the current health level in the user interface.
    - When the car's health reaches 0, it cannot be controlled at all, and the race is over.
    - When the car's health falls below 30%, the boost should not operate.

- Implement a first-person camera view. Use the "2" key to switch to this camera angle.
- Use the "1" key to switch back to the default chase camera.

- You must implement the collision detection for the tanks using **array or vector of tanks** and place your sphere-sphere collision detection algorithm in a **function**.

Upper second classification (60% +) Milestone 3 – break-even point achieved

- Resolve collisions with walls by manipulating the components of the momentum vector so the car appears to bounce off the walls.

- Introduce at least one further checkpoint. You will need to extend the number of states to account for the increased number of checkpoints. You also need to add appropriate dialogue.
    - Include some narrower walled sections that the car must travel through.

- Introduce a "Boost" facility to make the hover-car accelerate more quickly. This should be active while the space-bar is held down.
    - The thrust applied to the car should be 50% greater when the Boost is activated.
    - If the boost is active for too long (> 3 seconds), it will overheat. If this happens, the hover car should decelerate quickly (drag should be doubled), and the boost should not be usable for a period of time (5 seconds).
    - When the Boost is active, this should be shown on the dialog, with an additional warning 1 second before the booster overheats.

- Implement a non-player racing hover-car.
    - Use an array of dummy models to act as "waypoints" around the track which the non-player car should look at as it moves around the track.
    - As soon as it passes a waypoint (hint: use a distance calculation), the non-player car should look at and move towards the next waypoint.
    - The number and positioning of these waypoints will affect how realistic the movement looks
    - Use the same model for the non-player car, but edit a copy of the skin graphic so that it is a different colour.

- Implement collision detection and simple resolution between the player and non-player hover-car(s).

First classification (70% +) Milestone 4 – bonus payments received

- The hover-car should hover in the air as if it were on some sort of gravity cushion.
    - The car gently bobbles up and down as it moves.
    - Make the car "lean" into the bends as it turns.
    - The car should lift up slightly at the front or rear (you choose!) as it accelerates.
- The car should bounce when it collides with ***any*** object. The bounce needs to be smooth.

- Identify a successful move through a checkpoint. Place a cross centred under the checkpoint when the player has successfully negotiated that check-point. Provide it with a life timer so that the cross disappears after a short while.

- Make the race track into a complete circuit so that a race of more than one lap can take place. Show the current lap and total laps on the dialogue (e.g. Lap 2/5)
- Add a "current race position" to the dialogue, and at the end of the race, display the race winner and their race time.
- Give players the option to restart the race instead of quitting at the end of the game.

- The positions of all objects in the game should be read in from a simple text file, formatted as follows:
    - Each line of the file represents one scenic object, and has four fields separated by white space. The fields represent:
        - The type of object (text corresponding to the name of the .x file)
        - The x coordinate of the object
        - The z coordinate of the object
        - The rotation of the object around the y axis
    - The basic scene set-up (for 40%) would therefore look as follows:

```
Isle        -10  40   0
Isle         10  40   0
Isle        -10  56   0
Isle         10  56   0
Isle        -10  72   0
Isle         10  72   0
Wall        -10  48   0
Wall         10  48   0
Wall        -10  64   0
Wall         10  64   0
Checkpoint   0    0   0
Checkpoint  10  120  90
Checkpoint  25   56   0
```

High First classification (up to 100% and beyond!)

Only attempt the following section if you've already achieved the first classification:

These are suggested extensions – feel free to add anything extra that you want, as long as it doesn't break the requirements of the previous sections.
You do **not** need to attempt **all** of these to score 100%!

- Place some barrels round the track which contain burning fires
    - Implement the fire by using a particle system for the flames.
- Scatter some unexploded bombs (using the Flare model) round the track.
    - When either car passes too close to a bomb, it should explode
    - If a bomb explodes close to the player car, the camera should shake, and the explosion should cause damage to the car.
- Implement a particle system to simulate the exhaust flames of the Booster coming from the rear of the hover-car, when the boost is active.
- Use an array to act as a "speed table" for the non-player car, so that it behaves in a more realistic way.
    - The car would thus move at a customisable speed between each waypoint
    - You could make this smoother by having the car accelerate/decelerate between each new speed.
- Add more non-player hover-cars.
- Implement a damage model for the hover cars
    - Bumping into the hover cars (and causing them to bump into obstacles) will cause them to be damaged and slow down.
- Damaged cars could emit smoke, and/or list to one side
    - The player car, when damaged, could become more difficult to steer.
- Give the player car a limited range "photon torpedo" style weapon with which to attack and damage the non-player cars.
- Add a menu screen with the option to load different race tracks.

Not for marks, but if you really want to impress me…
This game is crying out for some development "tools" – primarily a level editor.
The level editor would help you create the sort of file described in the "first class" section. Develop it further, and you could release it as part of your game, to allow players to create and share their own race tracks.

Code Style and Layout

- Your code **MUST** be properly indented and laid out so that it is readable.
  - Brackets must line up (and should normally be on a line of their own).
  - Indentation must be consistent.
  - Appropriate use of white space should be made.
  - Over-long lines of code or comments should be split up
- You should have no "magic numbers" but instead make proper use of **constants**.
- **Variable names** must be meaningful.
- Your code should be **commented** appropriately.
- You should make proper use of **arrays / STL vectors**, with **loops** to process them:
  - **All** scenic items MUST be stored in arrays / STL vectors.
- You should make use of **enumerated types** to control the states within the game where appropriate.
- You MUST implement **functions** outside the main program as appropriate. I expect to see functions for collision detection and vector calculations as a minimum.
- You should make use of **structures** where appropriate.

Documentation

You need to produce a document to accompany your game.

The document will state:

- The **grade** you expect to get.
- The length of your hover-car in TL-Engine **units**.
- The length of your hover-car in **metres**.
- The **scale** of objects your game (units / metre)
- The coefficient of drag
- (for a 2:2 or higher) The maximum speed of your hover car (*without* boost) in
  - metres / second,
  - kilometres / hour
  - miles / hour
- (for a 2:1 or higher) The maximum speed of your hover car (*with* boost) in
  - metres / second,
  - kilometres / hour
  - miles / hour
- The maximum thrust force you apply (length of thrust vector) in TL-Engine units and scaled to represent acceleration in metres/second/second.

- A **brief** explanation of **How** you *resolve* collisions between your hover car and other objects.
  - This should include details the mathematics used for any calculations.

You must also include a **scale map** of your race course, showing the course boundary and the positions of obstacles, checkpoints, etc. (all labelled).
The positions of objects may be given in TL-Engine units, but the map should have a scale in metres.
The map should also state the length (in metres) of the race track (or circuit).